FSA Modernization Partner

United States Department of Education
Federal Student Aid

eZ-Audit Detailed Technical Design

Deliverable #86.2.2

Task Order #86

August 16, 2002

Document Revision History

Version No.	Date	Author	Revisions Made
1.0	8/1/2002	David Susanto	Created TOC.
1.0	8/16/2002	David Susanto	Created Overview Sections, incorporated first draft sections.
1.0	8/19/2002	Matt Portolese	Updated Sections 1.0 and 2.0.
1.0	8/19/2002	Brian Cannavan	Updated Section 2.3
1.0	8/19/2002	Andrew Smalera	Updated Section 3.0.
1.0	8/19/2002	Frank Soutfield	Updated Section 4.0
1.0	8/19/2002	David Susanto	Finalized document.

Table of Contents

1.0 Overview

1.1 Scope

The eZ-Audit Detailed Technical Design document describes the detailed Application Design, Application Architecture and Technical Architecture that are needed to develop eZ-Audit system. This document is created based on the functional design specified in the Use Cases specifications document and the high level technical architecture outlined in the Preliminary Technical Architecture Design document.

1.2 Intended Audience

This document is intended for individuals that have read and have the understanding of the eZ-Audit requirements, the Preliminary Design and the Functional Design documents (Use Cases). Basic understanding of Rational Unified Process (RUP) and Internet architectures / web application development and deployment concept and terminologies will help but not required to fully understand this document.

1.3 Application Design Overview

The Application Design section describes the User Interface (UI) design, Class and Sequence (application module) design and logical Data Model. The UI design is presented in the form of Low Fidelity (paper based) and High Fidelity (html based) screens. The Application Module design and the Logical Data Model are described by the sequence diagrams which contain the application modules and tables with their attributes.

1.4 Application Architecture Overview

The Application Architecture section explains the Web Conversation Framework (Strut), the Application Services, Component Services and Application Security. The Struts Framework explains the Model-View-Controller (MVC) architecture and how it relates to the eZ-Audit system. The Application Services describe how workflow and document management services are leveraged for the system. The Component Services explain how the system uses the Reusable Component Services (RCS) provided by the Mod Partner Integrated Technical Architecture (ITA) team. The Application Security architecture provides the information how the security and access level is going to be implemented at the application level.

1.5 Technical Architecture Overview

The Technical Architecture section describes eZ-Audit Development Architecture, Integration Architecture and Execution Architecture. The Development Architecture provides the detailed information of the Configuration Management and the Development Environment for the system. The Integration Architecture shows how the system interfaces with PEPS. The Execution Architecture describes the eZ-Audit customized architecture services that are needed during run-time such as Database Services, Document Management Services and Reporting Services.

2.0 Application Design

This section describes the detailed application design of the eZ-Audit system represented by the User Interface (Screen) design, Class (module) design and Data Model. This section will concentrate on the elements that essentially make up the eZ-Audit system at the application level.

2.1 User Interface Design

The User Interface (UI) Design is the look-and-feel representation of the eZ-Audit system. The UI design consists of Low Fidelity and High Fidelity design screens.

2.1.1 Low Fidelity Design

Low Fidelity Design is a UI design representation in a simple media such as Microsoft PowerPoint. The purpose of this design is to represent only the elements (hyperlinks, text, and form fields) that will appear on each web page in the system. The use cases were the basis for this representation, which were jointly designed by the project team (Accenture and core FSA team members). Page Flow Diagrams are also included as part of the Low Fidelity Design to show how the screens communicate with each other. The eZ-Audit Low Fidelity screens can be viewed in *Appendix A – Low Fidelity Design.* This document is located on eProject under directory *Deliverables/86.2.2 eZ-Audit Detailed Tech Design*. The Page Flow Diagrams can be viewed in *Appendix B – Page flow Diagrams*.

2.1.2 High Fidelity Design

High Fidelity Design is the actual look of the pages as they will be displayed on the web and is done in an HTML format. This design is based on the Low Fidelity Design but adds graphical pictures, elements and styles to the original content. The High Fidelity design will not function as a series of linked web pages such as a prototype, but was created as a group of standalone pages. Each HTML page is intended for development purposes and represents one of the final Java Server Pages or JSPs that will be created. The eZ-Audit High Fidelity screens can be viewed in *Appendix C – High Fidelity Design*.

2.2 Sequence and Class Design

The Sequence and Class Design shows the future eZ-Audit application modules and how the modules communicate within each JSP screen using a series of diagrams. Each diagram consists of a grouping of classes or objects (indicated by the boxes across the top) and the communication that takes place between the objects (represented by the arrows pointing from one line to another). The timing of the sequence begins with the first call numbered 1, and continues down the diagram. Each diagram only represents the communication that takes place between each of the classes and not the internal processing of each class.

Creating the diagrams portrays both the classes and business objects that need to be built. By creating the diagrams of each of the JSPs in the system, common business functionality can be identified, allowing the developers to reuse functionality and save development time. Common functionality in the "Action" classes can be found in the overview sequence diagram. These are calls that all modules will make and have been eliminated from each diagram for the purpose of simplicity. By combining the functionality found in the use cases for each module and the sequence diagrams, which show the necessary classes and the relationships between them, the developers will be able to code the system. The eZ-Audit Class and Sequence Diagrams can be viewed in *Appendix D – Sequence and Class Design*.

2.3 Data Model

The Data Model shows the tables, their attributes and the relationships among the tables that are needed to store data for the eZ-Audit system. The "Attributes" are individual data elements that need to be captured and saved in the application. The "Tables" are logical groupings of the attributes. The "Relationships" are the way these tables groupings of data attributes relate to each other. In order to accurately create this model, an intensive review of the use cases and low fidelity screen shots of the application was conducted for the sole purpose of determining what data was being created, used, and saved throughout the application. The finished

Data Model, as well as a logical breakdown of the model based on tables and relationships can be viewed in *Appendix E – Data Model*.

3.0 Application Architecture

eZ-Audit Application Architecture is the framework at the application level to provide the core application structure and common services for the eZ-Audit system. This section will concentrate only for the application architecture parts that require configuration or customization to serve the purpose of building the system.

3.1 Application Environment

The application environment is the set of core application architecture software components on which the custom development will be built. It is the foundation of software services that the system relies on. The core environment components are noted here because all major application design decisions have a dependency on them. For the eZ-Audit web application the core application environment is defined by the application server, IBM Websphere, and the database server, Oracle 8i.

3.1.1 IBM Websphere (3.5.3)

The IBM Websphere Application Server 3.5.3 (WAS) will be the deployment platform for the eZ-Audit web application. It provides the Java 2 Enterprise Edition (J2EE) standards support required for the eZ-Audit application architecture, as well as a scalable and robust run-time environment that will enable the application to meet its reliability and performance requirements. It is also the web application environment of choice for FSA. J2EE standards support that WAS provides includes:

- JDK 1.2.2 compliance
- JSP 1.1 specification compliance
- Servlet 2.2 specification compliance
- Servlet 2.1 Session API
- WAR file support for deployment
- JDBC 2.0 support (including connection pooling)
- JNDI 1.2 support

In addition, the IBM HTTP Server (IHS) and WAS together provide for secure HTTP communications via Secure Sockets Layer (SSL) 3.0.

3.1.2 Oracle 8i (8.1.7)

Oracle 8i (8.1.7) will be the database platform for the eZ-Audit web application. It is an industrial strength database server, with advanced reliability and performance capabilities. It has wide support for administration, tuning, data backup, and monitoring. It provides extensions to write database specific code in Java and PL/SQL, as necessary. It is also the database environment of choice for FSA. Oracle 8i integrates well with Websphere, through JDBC, for an optimal application and database environment.

3.2 Web Conversation Framework (Struts)

The purpose of the ITA Web Conversation framework is to provide a standard for developers and designers to apply the MVC design pattern for developing web applications. This framework will allow different tiers of the web application to be created independently of one another, provide the ability to update the static strings displayed without updating and recompiling code, and facilitates internationalization of web pages

3.2.1 Model-View-Controller Architecture Overview

A common problem that must be addressed in any User Interface (UI) systems development is how to handle all of the different types of processing going on. For example, there is the interaction with the user, display of information to the user, processing of business logic, and storage/update of static information (usually to a database). These activities constitute a minimum of what a UI application developer must consider. In order to handle these complex interactions, a framework or paradigm for development is necessary.
The MVC paradigm is a way of breaking an application, or even just a piece of an application's, into three parts: the model, the view, and the controller. MVC was originally developed to map the traditional input, processing, output roles into the Graphical User Interface (GUI) realm:

- Input --> Processing --> Output
- Controller --> Model --> View

The MVC design pattern separates business logic/state (the Model), from the User Interface (the View) and the program progression/flow (the Control). The user is presented with the View, which in a web-based application can be an HTML page or an image. The View interacts with the Controller, which is responsible for the flow of the program and processes updates from the Model to the View and vice versa. The Model represents the business logic or state (data in the system) and is usually defined as a Java object.

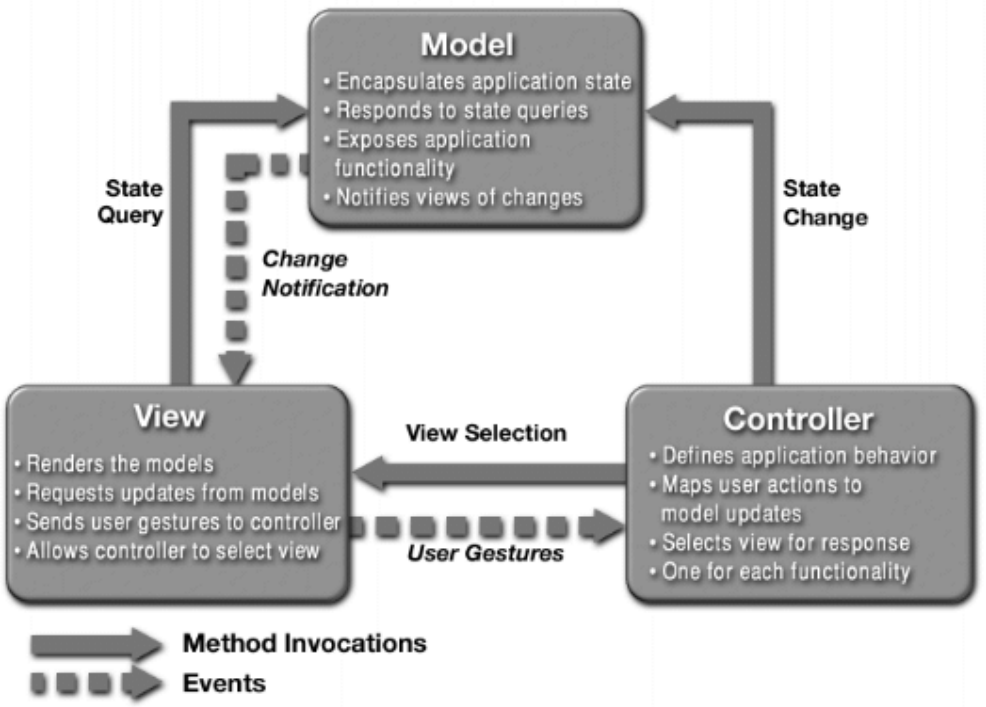


Figure 1. This diagram illustrates the general Model View Controller paradigm, on which Struts is based.

3.2.2 Struts Components

Struts is a set of cooperating classes, servlets, and JSP tags that make up a reusable MVC 2 design. This definition implies that Struts is a framework, rather than a library, but Struts also contains an extensive tag library and utility classes that work independently of the framework.
The figures below provide a good overview of the functionality the Struts components provide.

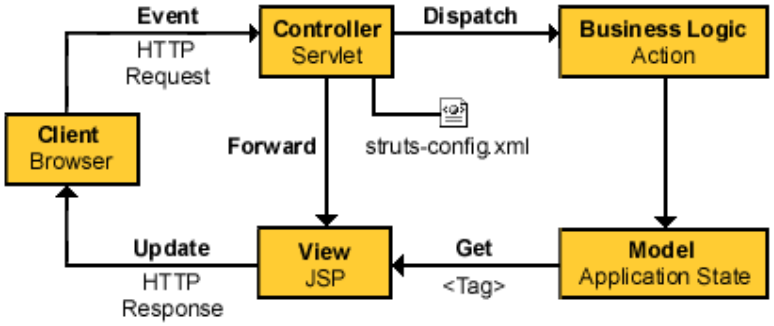


Figure 2. This figure displays the general functionality provided by the Struts framework.

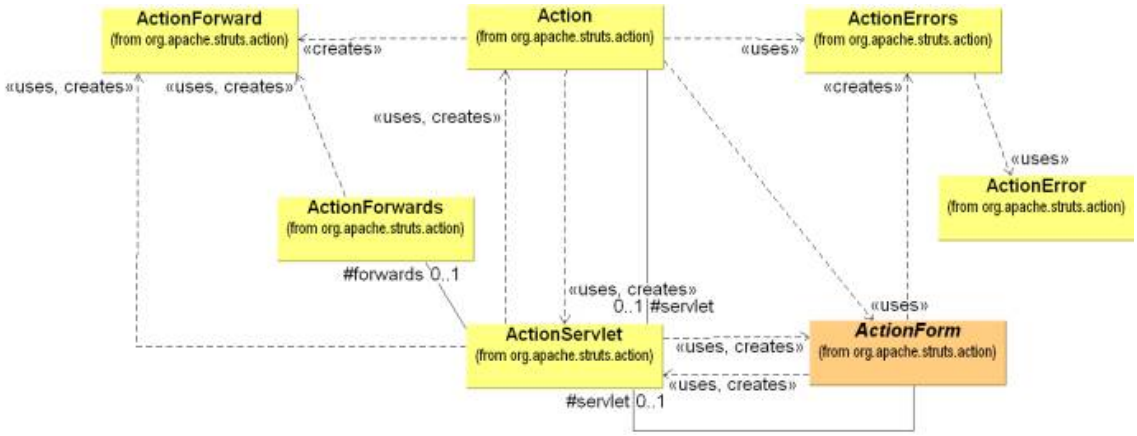


Figure 3. This figure displays the main Struts classes and how they interact. This diagram will be useful for reference in the discussion below.

There are several benefits to using Struts as the basis of the Web Conversation framework:

- Full separation of application flow, business objects, and presentation documents
- An existing web application structure that enables developers to focus on development of the application rather than the flow logic
- Simplified handling of user-provided form data
- Built-in configurable support for internationalization

3.2.3 Model Components

The **model** represents enterprise data and the business rules that govern access to and updates of this data. Often the model serves as a software approximation to a real-world process, so simple real-world modeling techniques apply when defining the model.

There are two main Struts model components. The first Struts model component is the **ActionForm** class**,** which maintains the session state for the Web application. ActionForm is an abstract class that is sub-classed for each input form model. When I say input form model, I am saying ActionForm represents a general concept of data that is set or updated by a HTML form. For instance, you may have a UserActionForm that is set by an HTML Form. The Struts framework will:

- Check to see if a UserActionForm exists; if not, it will create an instance of the class.
- Struts will set the state of the UserActionForm using corresponding fields from the HttpServletRequest. No more dreadful request.getParameter() calls. For instance, the Struts framework will take fname from request stream and call UserActionForm.setFname().

- The Struts framework updates the state of the UserActionForm before passing it to the business wrapper UserAction.
- Before passing it to the Action class, Struts will also conduct form state validation by calling the validation() method on UserActionForm. Note: This is not always wise to do. There might be ways of using UserActionForm in other pages or business objects, where the validation might be different. Validation of the state might be better in the UserAction class.

The UserActionForm can be maintained at a session level, and the struts-config.xml file controls which HTML form request maps to which ActionForm.

The second major Struts model component is the **Action** class, which is a wrapper around the business logic. The purpose of Action class is to translate the HttpServletRequest to the business logic. To use Action, subclass and overwrite the process() method.

The ActionServlet (Controller) passes the parameterized classes to ActionForm using the perform() method. This eliminates the need for repeated request.getParameter() calls. By the time the event gets here, the input form data (or HTML form data) has already been translated out of the request stream and into an ActionForm class.

3.2.4 View Components

A **view** renders the contents of a model. It accesses enterprise data through the model and specifies how that data should be presented. It is the view's responsibility to maintain consistency in its presentation when the model changes. This can be achieved by using a *push* model, where the view registers itself with the model for change notifications, or a *pull* model, where the view is responsible for calling the model when it needs to retrieve the most current data.

3.2.5 Controller Components

A **controller** translates interactions with the view into actions to be performed by the model. In a stand-alone GUI client, user interactions could be button clicks or menu selections, whereas in a Web application, they appear as GET and POST HTTP requests. The actions performed by the model include activating business processes or changing the state of the model. Based on the user interactions and the outcome of the model actions, the controller responds by selecting an appropriate view.

In Struts, the **ActionServlet** is the controller part of the MVC implementation and is the core of the Framework. ActionServlet (Command) creates and uses Action, an ActionForm, and ActionForward. As mentioned earlier, the struts-config.xml file configures the Command. During the creation of the Web project, Action and ActionForm are extended to solve the specific problem space. The file struts-config.xml instructs ActionServlet on how to use the extended classes. There are several advantages to this approach:

- The entire logical flow of the application is in a hierarchical text file. This makes it easier to view and understand, especially with large applications.
- The page designer does not have to wade through Java code to understand the flow of the application.
- The Java developer does not need to recompile code when making flow changes.

Command functionality can be added by extending ActionServlet.

3.3 Application Services

Application services consist of those pieces of functionality that can be considered as discrete architectural components, separate from the main application. They are discrete in the sense that they can be built as an independent set of software components (in this case JavaBeans and Tag Libraries), where they are general enough that they can be pulled out and re-used across applications. They also centralize the processing for

certain functional areas (in this case workflow and document management) into these subsystems. These services differ from the Component Services (see next section) only in that they are not fully packaged as reusable components, but they are good candidates to join the ranks of Component Services in the future, with some additional refinement.

3.3.1 Workflow Management

The workflow management process in eZ-Audit is straightforward; it requires keeping track of the state a submission is in, and controlling user interaction with the data on the submission depending on that state. The states a submission can be in are:

- In initial submission process (data is currently being entered by the Institution)
- Under review by the Screener
- Pending assignment to an Analyst by the the Co-Team Lead
- Under review by the Audit Resolution Specialist or the Financial Analyst
- Pending final review by the Co-Team Lead
- Closed

3.3.1.1 Architecture

In the eZ-Audit application, workflow management will be handled on two levels,

- Business objects
- User interface (UI)

Business objects will interact with the database to determine what submission belongs in which queue. In other words, when an institution has completed its submission, it will automatically then appears in the queue of the appropriate Screener. When the screener has checked off on a submission, it will appear in the queue of the appropriate Co-Team lead, and so on. Examples of business objects and their actions are available under the "Application Design – Sequence Diagrams" section of this document.

The user interface layer will control the actions that can be performed on a submission, depending on what workflow state it is in. That is, the display of the UI for each user will be tailored to allow only the operations that are valid for that user. For example, once a submission has been handed off by the school, the data on the forms and in the submitted attachments cannot be changed. Mechanisms for handling UI display are described below, under "Components".

3.3.1.2 Components

The eZ-Audit workflow framework will consist of a set of tag libraries (with processing in server-side Java classes) that provide information on the current submission's workflow state. The operations they perform are very straightforward and are described below:

- <ezaudit:displayIfInState
  docType="[AU or FS]" submitId="[sub_id]" state="[role_list]">
  [content]</ezaudit: displayIfInState >
  - this tag will send the [content] to the browser (typically HTML) if the state of the current (audit or financial statement) submission id appears in the comma-delimited workflow state list
- <ezaudit:displayIfNotInState
  docType="[AU or FS]" submitId="[sub_id]" state="[role_list]">
  [content]</ezaudit: displayIfNotInState >

- this tag will send the [content] to the browser (typically HTML) only if the state of the current (audit or financial statement) submission id does not appear in the comma-delimited workflow state list

- <ezaudit:processIfInState
  docType="[AU or FS]" submissionId="[sub_id]" state="[role_list]">
  [content]</ezaudit: processIfInState >
  - this tag will continue to process the [content] on the server (typically Java or JSP custom tags) if the state of the current (audit or financial statement) submission id appears in the comma-delimited workflow state list

- <ezaudit:processIfNotInState
  docType="[AU or FS]" submitId="[sub_id]" state="[role_list]">
  [content]</ezaudit: processIfNotInState >
  - this tag will continue to process the [content] on the server (typically Java or JSP custom tags) if the state of the current (audit or financial statement) submission id does not appear in the comma-delimited workflow state list.

3.3.2 Document Management

EZ-Audit will employ a document management solution to handle the storage and retrieval of documents related to the audit process. The details on the document management internals will be covered under the "Technical Architecture" section of this document. However, the application developers will need to interact with the document management solution and access its functionality. The following section describes the components that will be built to modularize the document management features such that they may be accessed by any JSP developer. Access to the document management API will be abstracted to its most basic components: how to store a document, and how to retrieve one.

3.3.2.1 Architecture

In the eZ-Audit application, document management will be handled on two levels,

- Business objects
- User interface (UI)

Business objects will interact with the database to store documents into and retrieve them from the data store . All of the implementation details of how the documents are to be physically stored and retrieved, and how they are catalogued and connected to a submission, will be encapsulated into the document management business objects, and accessed behind a standard set of API calls.

The user interface layer will handle displaying the links by which a user can access a stored document, or provide a document for uploading. UI components will bridge the gap between the business object calls, and the familiar HTML-based links that the user will see. Mechanisms for handling UI display are described below, under "Components".

3.3.2.2 Components

The eZ-Audit document management framework will consist of a set of tag libraries (with processing in server-side Java classes) that will provide UI functionality for storing and accessing documents. The operations they perform are very straightforward and are described below:

- <ezaudit:displayDocumentList submitId="[sub_id]">
  [content]</ezaudit:displayDocumentList>
  - this tag will send HTML links to the browser (typically HTML) that provide access to the

currently loaded documents for the provided submission ID. In between each link, the tag will return the HTML markup in the [content] section (if any), which will be useful when constructing the lists in an HTML table, for example.

- <ezaudit:displayDocumentUploadBox submitId="[sub_id]">
  [content]</ezaudit: displayIfInState >
  - this tag will send the HTML form markup for dialog boxes that allow for document uploading, according to how the document needs to be handled in the system

3.4 Component Services

Component Services are pre-built and packaged software pieces that eZ-Audit will leverage to minimize new development and eliminate 'reinventing the wheel'. These services are for the most part provided by the Integrated Technical Architecture (ITA) initiative, and have been designed, developed, tested, and packaged specifically for the purpose of reuse across multiple applications. Many valuable and time-consuming development tasks are avoided by the use of these components, such as logging, exception handling, and persistence. Details on the services are provided below.

3.4.1 Session Management

The session framework provides a mechanism for the retrieval and manipulation of user session and context data stored in cookies, web server session variables, or in a data store. The framework also creates one common interface for application developers to access the session via any of these methods.

The session framework will provide context management service that stores a user's temporary data during their HTTP session. The session framework will provide for the following services on both the client-side and server-side contexts:

- A common interface to all HTTP variables (request, cookie, session)
- Storing temporary data that should persist across each web page presented to the user

3.4.1.1 Components

The ContextManager class is the main class that application developers will call to obtain the session information. This class will require the calling application to provide the HttpServletRequest object and the type of method (client or server) to use to store session information.

The CookieRetrieval and SessionRetrieval are internal classes and are called by the ContextManager to access and manipulate the client and server session data.

3.4.1.2 API Information

The API Documentation for the ITA Session Framework can be found in the ITA RCS design document for this service. As the documentation is readily available in the design document and users guide, it is referenced here rather than reprinted.

3.4.2 Logging

The ITA Logging framework enhances the current logging ability of the ITA Web Application servers (specifically WebSphere), by allowing programmers to dynamically set logging and tracing functionality without modifying tested source code. The Logging Framework also allows log formatting standards to be developed and enforced to ensure proper information is gathered if a failure occurs.

The ITA logging framework is not tied to any specific application server. The ITA logging framework provides the following features:

- Simple logging API

- Background logging (*Configurable*)
- Multiple message severities
- Logs the name of the thread that issued the message (*Configurable*)
- Logs the name of the host that issued the message (*Configurable*)
- Arbitrary log channel names
- Configuration can be modified on-the-fly while the system is running

A running configuration can be written to XML for re-load later

3.4.2.1 Components

The Logging framework allows the programmer to log messages easily through a simple API. It contains one main object called Syslog. The Syslog object provides a flexible set of calls to log in a variety of manners. Most of the operation of Syslog is set through the XML configuration file the component uses. This makes re-configuration of the service very simple. The Logging framework can be used as a debugging tool during development and as a troubleshooting tool when the application goes into production. During development and testing, the messages may be simply sent to the console where the application is running or to a single log file. When the system is moved into production, log messages can be split up by severity levels (Fatal messages may result in someone being paged, for instance) and log files may be rotated every night and archived. These kinds of configuration changes do not require changes to the code and can even be made while the system is running. Detailed information on the logging framework can be found on the Logging design document.

3.4.2.2 API Information

The API Documentation for the ITA Logging Framework can be found in the ITA RCS design document for this service. As the documentation is readily available in the design document and users guide, it is referenced here rather than reprinted.

3.4.3 Exception Handling

The ITA Exception Handling framework is designed to provide a mechanism for trapping and dealing with any erroneous or unexpected actions that applications may encounter during runtime. The Exception Handling Framework also allows error message formatting standards to be developed and enforced to ensure proper information is gathered if a program failure occurs.

3.4.3.1 Components

The ITA Exception Handling Framework enhances the ability of an application to define and trap errors that may arise as the application executes. The Exception Handling framework includes the following key components:

- SFAExceptionFactory
- SFAException

The SFAException class is the base exception, which can be customized on an application by application basis. It has set() and get() methods for each of its main properties, such as errorCode, methodName, className, and so forth.

The SFAExceptionFactory is the class used to create exceptions. It should work in any application without modifications. This class is designed to work as a singleton (i.e. there is only one instance of it per VM or at least per classloader). Therefore, a reference to the factory is obtained via the getInstance() method. It also contains some exception-centered utility methods, such as getNestedException(), which can be used to obtain the nested exception for all standard Java exception types.

3.4.3.2 API Information

The API Documentation for the ITA Exception Handling Framework can be found in the ITA RCS design document for this service. As the documentation is readily available in the design document and users guide, it is referenced here rather than reprinted.

3.4.4 Persistence

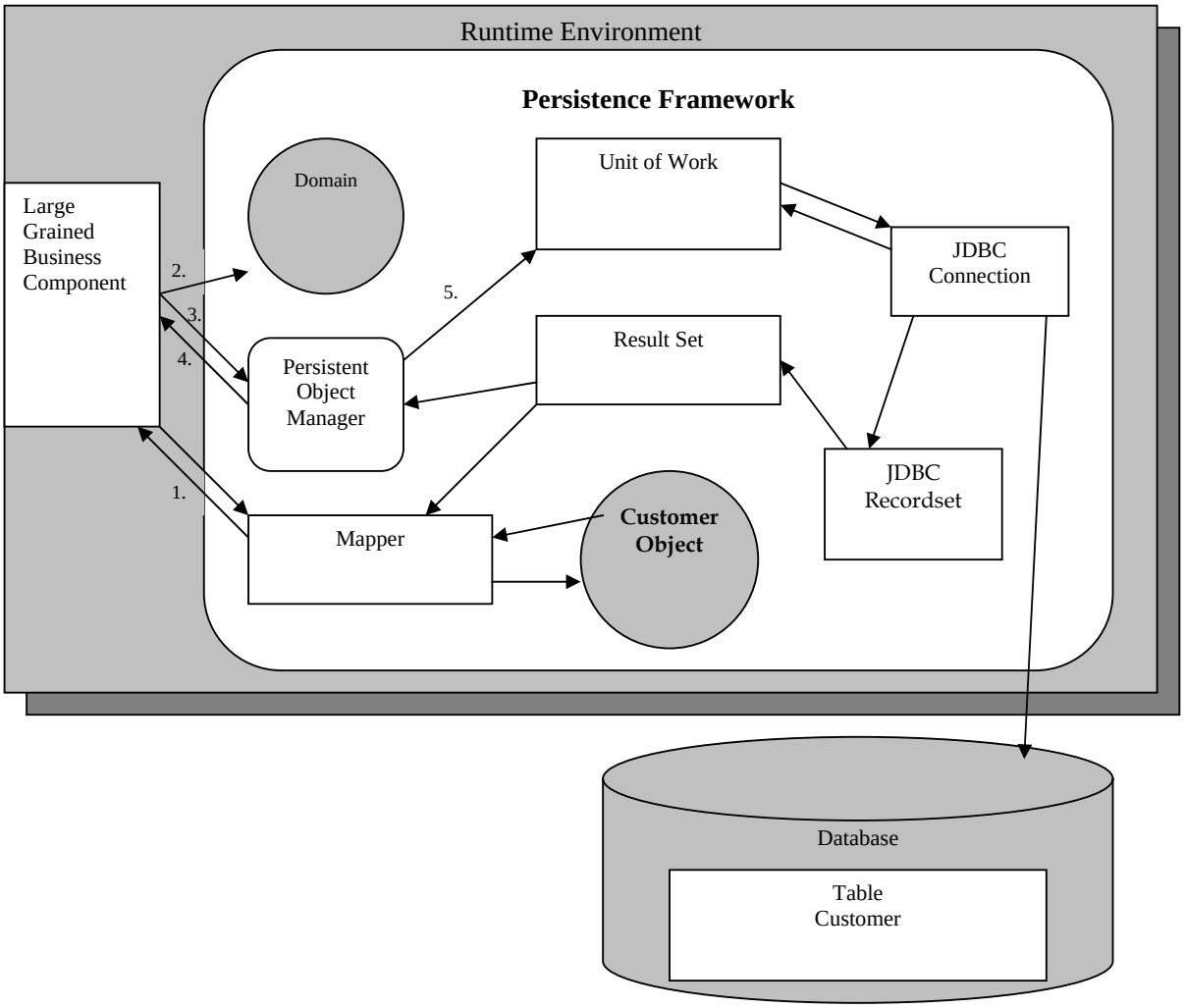
The ITA persistence framework provides a transparent and flexible mapping of the business objects to relational database tables. It is transparent in that once the business objects and their mappings are defined, application developers do not need to have any knowledge of the underlying relational database tables. It is flexible in that if the underlying relational database model changes, the business object model does not have to change with it – a change in the mapping layer is all that should be required.

3.4.4.1 Components

The framework is made up of several components working together:

- Domain Component
- Unit of Work Component
- Persistable Object Manager Component
- Result Set Component
- Business Mapper Component
- Business Object Component

A diagram of the components working together (retrieving a Customer Object from the database) is presented below:



Explanation of Steps:

1. Create Mapper and Business Object, populate values to update
2. Create Domain with connection information
3. Create Persistable Object Manager passing the domain
4. Request object from the POM passing the Mapper information
5. POM executes the query, maps the data to the object, closes the unit of work, and returns the object

3.4.4.2 API Information

The API Documentation for the ITA Persistence Framework can be found in the ITA RCS design document for this service. As the documentation is readily available in the design document and users guide, it is referenced here rather than reprinted.

3.4.5 JSP Tag Library Framework

The tag library framework provides a collection of commonly used JSP custom tag libraries for JSP developers to access. The JSP Tag Library framework is comprised of libraries leveraged from the Jakarta Struts framework, Apache Taglibs project, and custom developed libraries.

3.4.5.1 Components

The following is a list of JSP tag libraries provided in this framework:

- **Jakarta Struts Bean Taglib** - contains custom JSP tags used to define new beans from a variety of sources and to render bean or bean property output response
- **Jakarta Struts HTML Taglib** - contains JSP custom tags useful in creating dynamic HTML user interfaces, including input forms
- **Jakarta Struts Logic Taglib** – contains tags useful in managing conditional generation of output text, looping over object collections for repetitive generation of output text, and application flow management
- **Jakarta Struts Template Taglib** - contains tags that are useful in creating dynamic JSP templates for pages that share a common format
- **Jakarta DateTime Taglib** - contains tags that can be used to handle date and time related functions
- **Jakarta I18N Taglib** - contains tags that help manage the complexity of creating multi-lingual web applications
- **Jakarta Input Taglib** – contains tags that present HTML <form> elements tied to the ServletRequest. Can be used to pre-populate form elements with prior values that the user has chosen or with default values
- **Log Taglib** – is used to embed logging calls in JSP using the Logging Framework
- **Jakarta Page Taglib** – contains tags that can be used to access all of the PageContext information of a JSP page provided 'page'attributes
- **Jakarta XSL Taglib** – contains tags used to process an XML document with an XSL stylesheet and incorporate the data in the page
- **Jakarta XTags Taglib** – contains custom tags for working with XML and implements an XSLT-like language allowing XML to be styled and processed from directly within a JSP. XTags is currently built on DOM4J foundation, an open source XML framework for the Java platform.

3.4.5.2 API Information

The API Documentation for the ITA JSP Tag Library Framework can be found in the ITA RCS design document for this service. As the documentation is readily available in the design document and users guide, it is referenced here rather than reprinted.

3.4.6 Configuration

The configuration framework allows configuration data to be stored in the form of properties files, xml files, database tables, or any combination of the three. The framework creates one common interface for application developers.

The Configuration Framework provides the following services:

- Configuration Data Load: the config data files are loaded into static
- Configuration Data Retrieval: returns the value based on the domain name and the tag/key name.

3.4.6.1 Components

The **FSAConfiguration** class is the class that contains the static initializer. It will find and load the master domain data. It is the concrete facility class that is used to retrieve configuration data from a variety of sources. This class's primary responsibility is to supply configuration data from a variety of sources in a uniform fashion.

The **FSAXmlResourceBundle** class extends the java.util.ResourceBundle class. This class locates XML files within the CLASSPATH.

The **GrndsConfigurationEnvironment** class extends the java.util.Properties class. This class encapsulates two ideas in application configuration: properties and configuration classes. This class bridges both approaches to representing configuration information by extending the java.util.Properties class and offering the ability to get the configuration objects created from all sources.

The **GrndsConfigurationSource** interface defines the base interface that all concrete sources must implement. The GrndsSystemPropertySource class, GrndsPropertyFileSource class, GrndsXmlFileSource class, and GrndsDatabaseSource class extend this class.

3.4.6.2 API Information

The API Documentation for the ITA Configuration Framework can be found in the ITA RCS design document for this service. As the documentation is readily available in the design document and users guide, it is referenced here rather than reprinted.

3.5 Application Security

The primary objective for application security in eZ-Audit is to enforce the appropriate data interaction policy for a given user. That is, the application will control, based on the user's permissions, the ability to create/read/update/delete a given piece of data. The set of data interaction permissions listed here is commonly known as a CRUD (Create/ Read/ Upate/ Delete) matrix.

The user's CRUD permissions will be enforced through the user interface. That is, the display of the UI for each user will be tailored to allow only the operations that are valid for that user. Specifically:

- If a user should not have permission to create (C) a new record, the button or link for creating that record will not be displayed.
- If a user should not have permission to read (R) a record or a piece of data, the link to that data, or the data itself, will not be displayed.
- If a user should not have permission to update (U) a record, then the fields in that record will be displayed as read-only.
- If a user should not have permission to delete (D) a record, the button or link for deleting that record will not be displayed.

UI based security controls are very effective, because the user is never even allowed to start a process of reading or updating data when it is not allowed to him or her. This eliminates extra processing steps, pop-ups or warning messages after the fact, decreases user frustration, and improves the general usability of the application.

3.5.1 User Model (Roles and Permissions)

The eZ-Audit application will implement its security policy through the checking of roles for display and programmatic control. General categories of access rights are defined as security roles, and any user of the system can be defined as being in one of these categories or roles. A user can access a resource in the application only if his/her role has been granted that access. The list is as follows:

- ED Admin
- Institution Admin
- Case Team Admin
- Data Entry

- Screener
- Audit Specialist
- Financial Specialist
- Case Assignment
- Case Approval

3.5.2 Security Implementation Model

The security implementation model is the set of access control, administration and runtime components that will be utilized to enforce the security model as its requirements dictate. The sections below cover each of these components.

3.5.2.1 Access Control

Access control determinations will be made entirely in the user interface code. That is, the UI layer of the application will be solely responsible for displaying the data or widgets for a user to:

- Be able to navigate to a link, or a region of the application
- Be able to see the data in read-only format
- Be able to peform updates on the data
- Be able to create a new record in the database

The application JSP files will contain all of the UI HTML and then display the appropriate HTML conditionally for the appropriate role.

3.5.2.2 Administration

All administration of user accounts and roles will be performed in administration console screens of the eZ-Audit application. No additional screens or tools will be necessary.

3.5.2.3 Run-time Components

The eZ-Audit security framework will consist of a set of tag libraries (with processing in server-side Java classes) that provide information on the current user's role. The operations they perform are very straightforward and are described below:

- <ezaudit:displayIfInRole roleName="[role_list]">[content]</ezaudit: displayIfInRole >
  - this tag will send the [content] to the browser (typically HTML) if the role of the current user appears in the comma-delimited role list
- <ezaudit:displayIfNotInRole roleName="[role_list]">[content]</ezaudit: displayIfNotInRole >
  - this tag will send the [content] to the browser (typically HTML) only if the role of the current user does <u>not</u> appear in the comma-delimited role list
- <ezaudit:processIfInRole roleName="[role_list]">[content]</ezaudit: processIfInRole >
  - this tag will continue to process the [content] on the server (typically Java or JSP custom tags) if the role of the current user appears in the comma-delimited role list
- <ezaudit:processIfNotInRole roleName="[role_list]">[content]</ezaudit: processIfNotInRole >
  - this tag will continue to process the [content] on the server (typically Java or JSP custom tags) if the role of the current user does <u>not</u> appear in the comma-delimited role list.

3.6 Application Architecture References

- Struts Homepage

  http://jakarta.apache.org/struts

- Struts Documentation - Apache Struts Framework (Version 1.0)

  http://jakarta.apache.org/struts/api-1.0/index.html

- Struts, an open-source MVC implementation

  http://www-106.ibm.com/developerworks/ibm/library/j-struts/

- Strut Your Stuff with JSP Tags: Use and extend the open source Struts JSP tag library

  http://www.javaworld.com/javaworld/jw-12-2000/jw-1201-struts.html

- Introduction to Jakarta Struts Framework – Parts 1 – 3

  http://www.onjava.com/lpt/a//onjava/2001/09/11/jsp_servlets.html
  http://www.onjava.com/pub/a/onjava/2001/10/31/struts2.html
  http://www.onjava.com/pub/a/onjava/2001/11/14/jsp_servlets.html

- The Struts Framework's Action Mappings Configuration File

  http://www.informit.com/content/index.asp?product_id={0917F29F-56D8-4B25-9C67-211EC945BBAB
  (Requires creating a free informit.com account.)

- Building a Web Application: Strut by Strut

  http://husted.com/about/scaffolding/strutByStrut.htm

- Java Developer's Journal

  http://www.sys-con.com/java/article.cfm?id=1175

- "Introduction to MVC and the Jakarta Struts Framework," Craig W. Tataryn

  http://www.computer-programmer.org/articles/struts/

- "**J2EE FrontEnd Technologies: A Programmer's Guide to Servlets, JavaServer Pages, and Enterprise JavaBeans, " Lennart Jörelid**

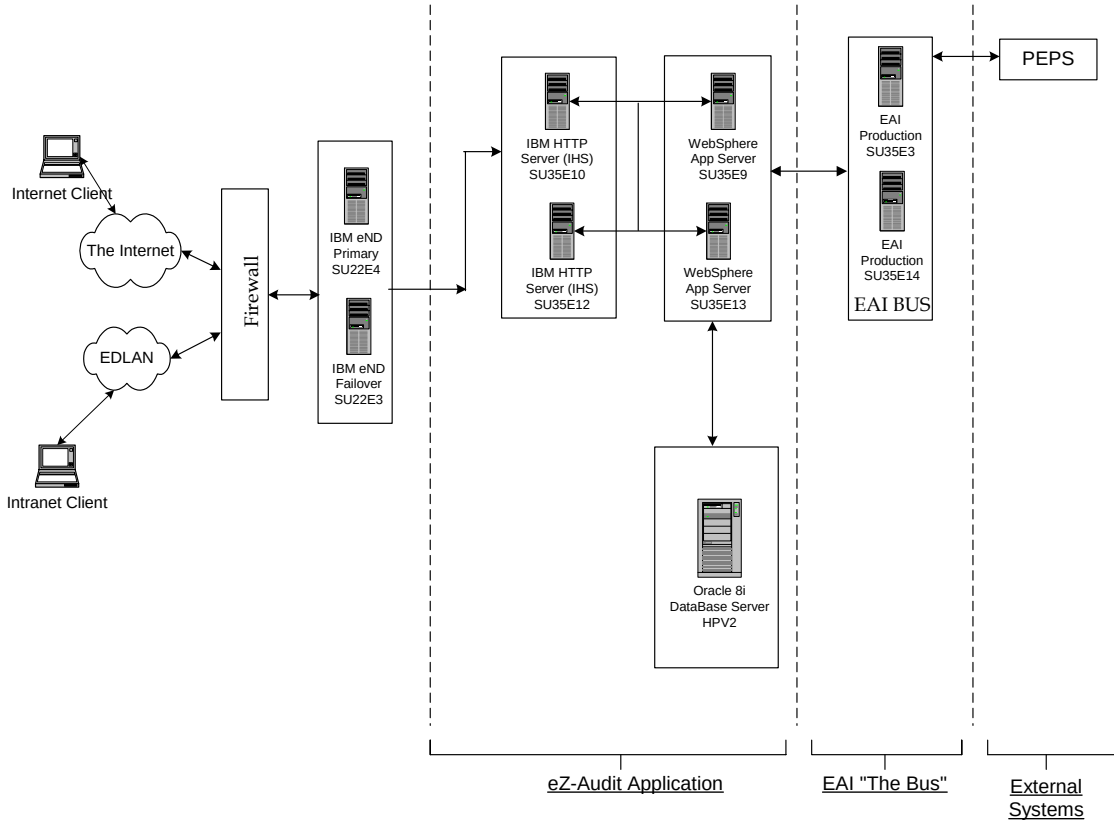
4.0 Technical Architecture

eZ-Audit Technical Architecture is the hardware and software foundation where the eZ-Audit system will be built. This section will concentrate only for the architecture parts that require configuration or customization to serve the purpose of building the system.

4.1 Technical Architecture Overview

The eZ-Audit system is a web-based application. Most of the system data will be resident in the eZ-Audit database; however, there are certain interface requirements for accessing data stored in external systems. At the core, eZ-Audit is comprised of the VDC network infrastructure and file system and the ITA server infrastructure.

The diagram below depicts the eZ-Audit production architecture. The only system being integrated in for Release 1.0 is PEPS.



Systems Architecture Diagram

The table below lists the System Architecture components and their relevance in the eZ-Audit architecture.

System Component	Product/FSA System	Provided By
Firewall	IBM Checkpoint	VDC
Load Balancer	IBM eND	VDC
Web Server	IBM HTTP Server	ITA
Application Server	IBM WebSphere Application Server Advanced Edition	ITA
Database	Oracle 8i	ITA / VDC
User Profile Database	Oracle 8i	EZ-Audit
Enterprise Application Integration (EAI)	IBM MQSeries	EAI
Document Storage & Imaging	VDC Storage Area Network	VDC
Non Financial & Audit School Data	PEPS	PEPS through EAI

4.2 Development Architecture

The eZ-Audit development architecture has two core components: configuration management and the development environment. Configuration management defines the process of managing the development of application code and releasing the code in a structured manner. The development environment is the set of hardware and software used to develop, test, and deploy the application. The following sections describe these areas in more detail.

4.2.1 Configuration Management

Rational ClearCase will be the software used for Configuration Management and Version Control throughout the eZ-Audit development phase. It is the standard tool used and supported by the ITA team. No other Configuration Management tools were considered for this reason.

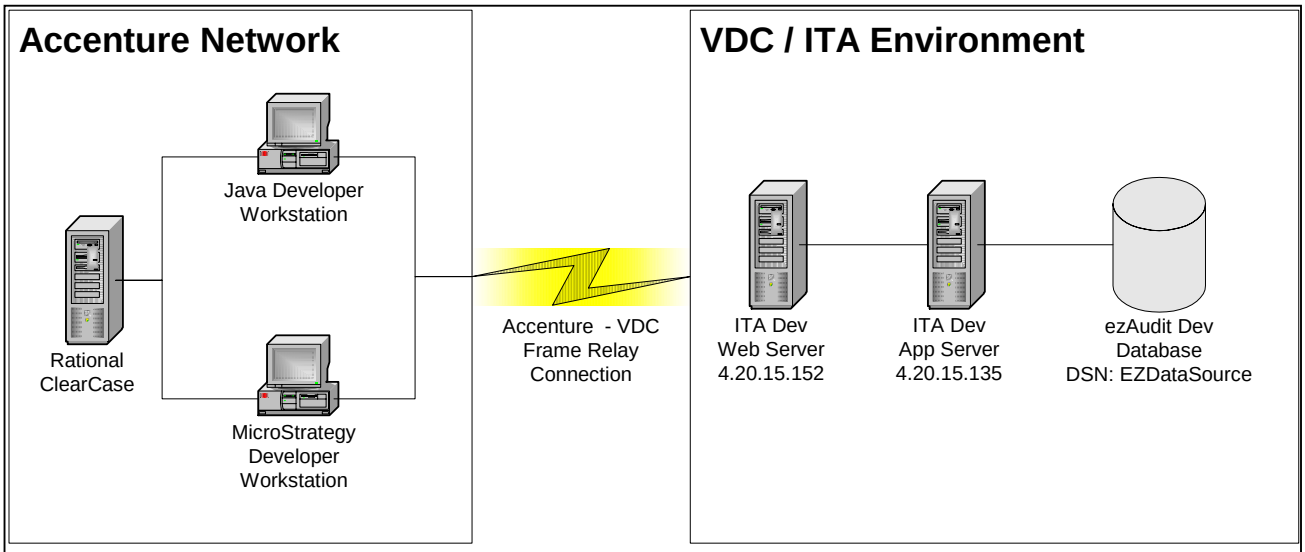
The ClearCase repository will be stored on the Accenture LAN. This is in place of housing the repository at the VDC. This decision was made so ClearCase will be administered by the eZ-Audit project team which eliminates dependency on the VDC for this service. Team members working from the EDLAN will not have access to the ClearCase repository. This is not a concern since the development team are the only users of ClearCase and will be working from the Accenture LAN.

4.2.2 Development Environment

The eZ-Audit development environment is composed of three primary components:

1. **ITA Development Environment** – Includes a web server, application server, RCS components, and database instance for development and unit testing. The development servers are hosted at the VDC.
2. **Rational ClearCase** – Used for version control and source code management of application code and configuration modules.
3. **Development workstations** – Primary development machines for application and report development with connections to both the ITA Development Environment at the VDC and the Rational ClearCase code repository.

The diagram below illustrates the development environment components.



Development Environment Components:

1. Java Developer Workstation - Primary development environment for developers. Used for development of the JSPs, Servlets, and Java Beans.
2. Microstrategy Developer Workstation - Development environment for creating and testing the Microstrategy reports.
3. Rational ClearCase - Version Control software used to check-in and out application and configuration modules during development. Will be used as the build source for testing.
4. Web Server - ITA provided web sever instance for ezAudit development.
5. Application Server - ITA provided application server instance for ezAudit development.
6. Database - Oracle 8i database instance for ezAudit development.

The following sections outline the specific configuration of the ITA development environment.

4.2.2.1 Server Information

Server	Machine Name	IP Address
Web Server	su35e2	4.20.14.132 (from EDLAN and Frame Relay use 4.20.15.132)
Application Server	su35e5	4.20.14.135 (from EDLAN and Frame Relay use 4.20.15.135)

4.2.3 Directory Structure

4.2.3.1 Web Server

Directory	Description
/www/dev/ezaudit/htdocs	static content document root, location of HTML files

4.2.3.2 Application Server

Directory	Description
/www/dev/ezaudit/<env>/servlets	location of servlets, classpath
/www/dev/ezaudit/<env>/web	dynamic content document root, location of JSPs
/www/dev/ezaudit/<env>/jars	location of jar files in classpath
/www/dev/ezaudit/<env>/logs	location of log files

<env> is a number 1, 2, 3 or 4. These numbers correspond to the four separate environments built. /www/dev/ezaudit/1/ is the root of http://dev.ezaudit.ed.gov:8531/EZ1WebApp/. "2" is the root for EZ2WebApp and so on.

4.2.3.3 URL Information

The development eZ-Audit web page is protected and requires basic authentication to access. Developers will have to modify their host file to reach the eZ-Audit development web page to include an entry for dev.ezaudit.ed.gov. For example, for a Windows NT workstation on Frame Relay find the host file (c:\winnt\system32\drivers\etc\host) and edit it to contain the line below:

4.20.15.132 dev.ezaudit.ed.gov

Save and exit the file.

The development eZ-Audit web page can be accessed at:

http://dev.ezaudit.ed.gov:8531/

All requests to WebSphere must use the web application name EZ<env>WebApp. See examples below:

http://dev.ezaudit.ed.gov:8531/EZ1WebApp/viewform.jsp
http://dev.ezaudit.ed.gov:8531/EZ2WebApp/viewform.jsp
http://dev.ezaudit.ed.gov:8531/EZ3WebApp/viewform.jsp
http://dev.ezaudit.ed.gov:8531/EZ4WebApp/viewform.jsp

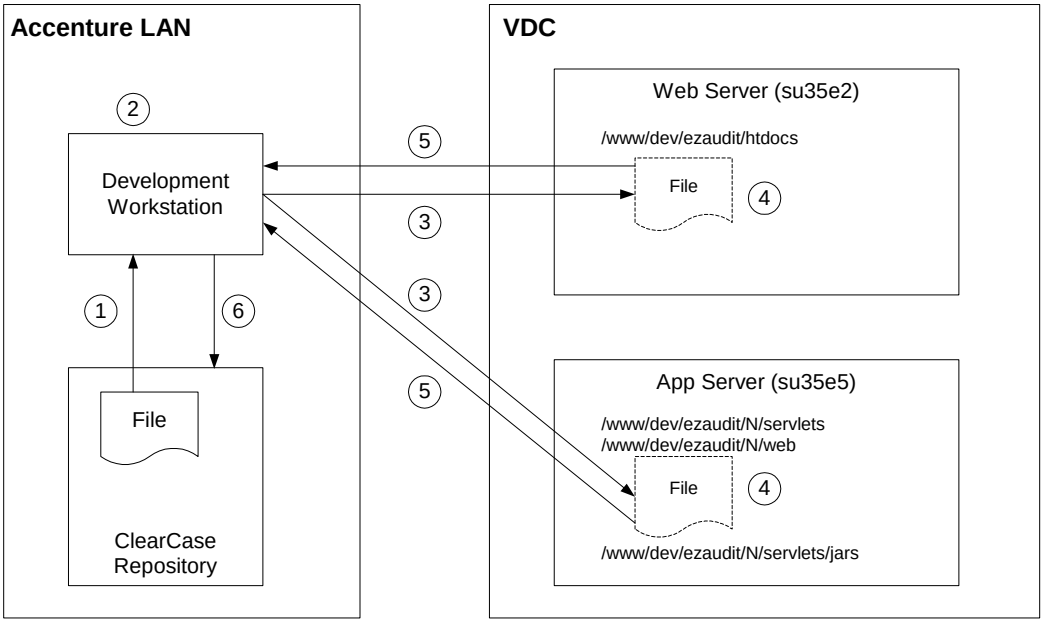
4.2.3.4 Database Information

The DataSource information is listed below:

DataSource Name: EZDataSource
JNDI Name: "jdbc/EZDataSource"
Description: Oracle development database: EZAUDEV

4.2.4 Development Process

The proposed development environment consists of individual developer workstations, a ClearCase repository, a web server, an application server, and a database server. The following diagram illustrates the typical procedures a developer will make when developing a static HTML page, JSP, servlet, or JavaBean. Please note that the "N" in the file path represents a number that corresponds to a particular development directory.



1) A file is checked out from the eZ-Audit ClearCase repository.
2) The developer makes changes or creates a new file on the development workstation.
3) The file is FTP'd from the workstation to the appropriate directory on either the Web Server or the Application Server.
4) The file is executed and/or tested once on the development servers. The file may also be changed here if necessary.
5) If the file is changed on the server it must be FTP'd back to the Development Workstation.
6) The file is checked back in to the ClearCase repository.

4.3 Integration Architecture (PEPS Integration)

EZ-Audit integrates with PEPS for school file information. The integration is a two-way integration, with eZ-Audit receiving updated school information daily from PEPS and PEPS having read-only access to eZ-Audit data for their application. The following sections describe each of these in more detail.

4.3.1 School File Feed

Ez-Audit receives a daily feed from PEPS for school related information. The school file contains the current information about a school including:

- OPEID
- School Name
- Reporting Status
- Region
- School Group
- State
- Etc.

Refer to the application design section for the complete set of school file data used by eZ-Audit.

This information is updated daily from PEPS and sent to eZ-Audit through the EAI Bus and placed in a directory accessible to eZ-Audit. There will be an eZ-Audit script to read the file and update the school information in the database. The PEPS feed is a complete set of PEPS information every day with an

indicator on whether a record has changed or not. The EAI Bus will send only the delta file to eZ-Audit for processing. The file will be loaded into the eZ-Audit database as a nightly batch process to minimize any impact on the application.

4.3.2 PEPS Access to eZ-Audit Data

The PEPS application requires access to eZ-Audit data for:

- Reporting (Consolidated and other)
- Forms / Screens pre-population
- Data Access for non-eZ-Audit users (OGC, OPE,Guaranty Agencies, State Licensors, Crediting Agencies, and other Dept. Of Ed Uses)

The PEPS application will have read-only access to the eZ-Audit tables to present the information required in the PEPS application. To facilitate the Consistent Data approach, data cannot be updated via PEPS and PEPS will not persist any information in its own database. PEPS will write any required logic to reference eZ-Audit data in a read-only manner.

4.4 Execution Architecture

The eZ-Audit execution architecture consists of Database Services, Document Management Services, and Report Services. The database services are provided for user authentication and profile management. Document management services ensure that all the information required for future Accorde integration is captured and part of the application design. Report services are actually handled by the portal applications with eZ-Audit data accessible through predefined reports. The following sections describe these areas in more detail.

4.4.1 Database Services

A Technical Architecture decision has been made to deviate from the original plan to use an LDAP-compliant directory for user authentication and authorization. This decision was made based on the benefits gained versus the cost of implementing an LDAP architecture.

The use of eZ-Audit would not realize the largest benefits of using LDAP. These include the ability to access the directory from any computing platform and from many locations such as in an enterprise-wide implementation. The LDAP proposed for eZ-Audit would only be used by eZ-Audit and not gain these benefits.

FSA does not have an LDAP standard and any implementation of an LDAP directory may necessitate changing products to meet a future standard. All functionality needs of using an LDAP database for eZ-Audit's purposes can be met by Oracle database supported authentication and authorization methods.

4.4.2 Document Management Services

As part of the eZ-Audit application, users will upload documents associated with a school. These documents will be stored by eZ-Audit and can be viewed by users, but not updated. Users have the ability to submit additional documentation associated with an audit or resubmit all documents, but not update individual documents. The following sections outline the location of where the documents will be stored, the naming conventions for the documents, the rules surrounding handling of old files, and the capture of meta-data information required for future ACCORDE integration / data feeds.

4.4.2.1 Document Storage

The documents will be stored on the Storage Area Network (SAN) at the VDC. The SAN is a RAID 5 disk array storage system that provides secure, redundant data storage to protect against a single drive failure causing the loss of data. The SAN will be a mapped drive to the eZ-Audit web servers.

The key feature of the SAN for eZ-Audit application is the ability to redundantly store data to prevent a single hardware failure from compromising audit data. The SAN is designed to provide 100% availability and allows drives to be swapped out without downtime or loss of data in the case of a drive failure.

4.4.2.2 File Naming

The files need to be named to uniquely identify the file both for the school / user submitting it and to ensure that the correct / latest version of the file is being referenced. The data elements that make a document unique are:

1. Date / Time stamp
2. OPE ID
3. Doc Category

Documents will be named with the following convention for storage:

OPEIDMMDDYYYYHHMMSS.pdf

The original file name that the document had will be stored in the database for display to the user. The new name is designed to ensure that files do not unintentionally overwrite files.

4.4.2.3 Handling of Old Files

When files are replaced, they will be changed to an inactive status. This will give traceability required for audits and ensure old copies of the documents are retained. A file will be moved to closed status after the 2 years have expired when the file must be kept. The file can then either be deleted or archived outside of the application.

4.4.2.4 ACCORDE Meta-Data Information

For Release 1.0, eZ-Audit will not integrate directly with the ACCORDE application. To facilitate future integration, the eZ-Audit application will ensure that all ACCORDE meta-data information is captured and associated with files uploaded for storage.

The database would need to hold the following information about a document:

ACCORDE Feed Name	EZ-Audit Field Name	Source	Description
ACN	Submission; ACN	eZ-Audit	Audit Control Number
BATCHNUMBER	N/A	N/A	ACCORDE batch ID. Not applicable at this point to eZ-Audit
CATEGORY		eZ-Audit	For eZ-Audit, the value will always be AUDREP
CONAME	N/A	N/A	An ECMC internal field.
COUNTRY	Institution;Country	eZ-Audit	Part of the PEPS to eZ-Audit feed.
DOCTYPE	File_Upload;Type	eZ-Audit	
FIELD1	N/A		Placeholder fields for future index data, could be submitted if there was a need.

ACCORDE Feed Name	EZ-Audit Field Name	Source	Description
FIELD2	N/A		Placeholder fields for future index data, could be submitted if there was a need.
FIELD3	N/A		Placeholder fields for future index data, could be submitted if there was a need.
LOB	N/A	N/A	ECMC internal field.
OPEIDNO	Institution;OPE_ID	eZ-Audit	OPEID is taken from the PEPS school file to identify the school.
ORIGDATE	File_Upload;OrigDate	eZ-Audit	Date when the school creates the file.
PRCN	N/A		Not applicable to eZ-Audit.
RECDATE	File_Upload;Upload_Date	eZ-Audit	Date when FSA receives the document. Should be the date when uploaded into the system.
REGION	Institution; Region_ID	eZ-Audit	Part of the PEPS to eZ-Audit feed.
SCHOOLGROUP	Institution; Group_Num	eZ-Audit	Part of the PEPS to eZ-Audit feed.
SCHOOLNAME	Institution; Name	eZ-Audit	Part of the PEPS to eZ-Audit feed.
SCHOOLYEAR	Institution; Fiscal_Year_End	eZ-Audit	Part of the PEPS to eZ-Audit feed.
STATE	Institution;State	eZ-Audit	Part of the PEPS to eZ-Audit feed.
UNIQUEID		ACCORDE	ACCORDE internal identifier. Not provided by eZ-Audit

The table below lists the additional fields related to the uploaded files and their purpose.

Field	EZ-Audit Table and Field Name	Description
Original File Name	File_Upload;Orig_File_Name	The file name that the user gave the file when it was created. This is maintained to display a common name. There is no way to ensure this filename is unique, so it is not used for storing the file.
Status	File_Upload;Status	Status indicates whether a file is active or not. A file may be turned to inactive if either the file is replaced or if the file has expired.

4.4.3 Report Services

Reports for eZ-Audit will be created with the MicroStrategy reporting application and accessed through the Data Mart applications. Users requiring access to reports will have an additional login id for the Data Mart sites and access the eZ-Audit reports through an eZ-Audit tab at the top of the screen.

The Data Mart sites will have access to real-time eZ-Audit data and will report on live data. The specific reports available can be found in the Reports Use Case and the application design section.